Apuntadores

La memoria de una PC puede verse como un vector de valores, donde cada una de sus posiciones es referenciada por un número hexadecimal llamado "dirección de memoria".

Los apuntadores son variables cuyo contenido es una dirección de memoria, que almacena la dirección de memoria de otras variables -int, float, char-, por lo que un apuntador apunta a la variables cuyo valor se

almacena a partir de la dirección de memoria de contiene.

Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

Los apuntadores solo pueden apuntar a direcciones de memoria del mismo tipo con el que fueron declarados.

Declaración de apuntadores:

```
int *a1; //a1 apunta a un valor de tipo entero
char *a2; //a2 apunta a un valor de tipo carácter
float *a3; //a3 apunta a un valor de tipo real
```

Recuerden que para declarar un apuntador se antepone un *.

Operadores unitarios en apuntadores:

& Para decir dirección de memoria.

***** Para decir contenido de la dirección de memoria.

Ejem1

```
#include<stdio.h>
int main(){
    int x=10, *a1, *a2;
    a1=&x;
    printf("\n%p", a1);
    a2=a1;
    printf("\n%p", a2);
    *a1=20;
    printf("\n%d", x);
return 0;
}
```

Es posible sumar y restar valores enteros a un apuntador. El resultado de estas operaciones es el desplazamiento de la dirección de memoria hacia adelante (suma) o hacia atrás (resta) por bloques de bytes del tamaño del tipo de dato apuntado por el apuntador. Esto permite recorrer arreglos utilizando apuntadores.

Ejem2

```
#include<stdio.h>
int main(){
    int x=10, *a1, *a2;
    a1=&x;
    printf("\n%p", a1);
    a2=a1+2;
    printf("\n%p", a2);
    *a1=20;
    printf("\n%d", x);
return 0;
}
```

Ejem3

```
#include<stdio.h>
int main(){
    int x=10, y=50, *a1, *a2;
    a1=&x;
    printf("\n%p", a1);
    a2=a1+1;
    printf("\n%p", a2);
    *a1=20;
    printf("\n%d", x);
    *a2=y;
    printf("\n%d", *a2);
return 0;  }
```

Ejem4

```
#include<stdio.h>
int main(){
    int x=10, y=50, *a1, *a2;
    a1=&x;
    printf("\n%p", &x);
    printf("\n%p", a1);
    a2=&y;
    printf("\n%p", &y);
    printf("\n%p", a2);
    *a1=20;
    printf("\n%d", *a1);
    printf("\n%d", x);
```

```
    *a2=100;
    printf("\n%d", *a2);
    printf("\n%d", y);
return 0;
}
```

Ejem5

```
#include<stdio.h>
int main(){
    int x=10, y=50, *a1, *a2;
    a1=&x;
    a2=&y;
    x=y;
    if(a1==a2)
        printf("\nSon iguales\n");
    else
        printf("\nHola\n");
return 0;}
```

Ejem6

```
#include<stdio.h>
int main(){
    int x=10, y=50, *a1, *a2;
    a1=&x;
    a2=&y;
    x=y;
    if(*a1==*a2)
       printf("\nSon iguales\n");
    else
       printf("\nHola\n");
return 0;}
```

Ejem7

```
#include<stdio.h>
int main(){
    int x=10, y=50, *a1, *a2;
    a1=&x;
    a2=&y;
    x=y;
    if(&x==&y)
        printf("\nSon iguales\n");
    else
        printf("\nHola\n");
return 0;}
```

Estructuras

Una estructura es una colección de una o más variables iguales o de distinto tipo, agrupadas bajo un mismo nombre, es decir un tipo de dato compuesto que permite almacenar un conjunto de datos de diferente tipo -agrupar un grupo de variables relacionadas entre si- que pueden ser tratadas como una unidad –bajo un mismo nombre se definen.-

Las estructuras pueden contener tipos simples y tipos compuestos.

Simples: variables de tipo carácter, entero y real.

Compuestos: variables tipo arreglos y estructuras.

Declaración de una estructura.

```
//Global
struct nombreEstructura{
    variablesInvulocradas;//declaración de variables
};
```

O bien

```
//Local
struct {
    variablesInvulocradas;//declaración de variables
}nombreEstructura;
```

Una vez definida una estructura global, es posible crear variables de ese tipo en la función que se use por ejemplo:

```
struct nombreEstructura nombreVariable;
```

Ejem8

```
#include<stdio.h>
#include<math.h>
#define p printf
#define s scanf

struct polar{ //global
      int a;
     float b;
 };
```

```
int main(){
  struct polar p1;
  float r, A;
  p("\t **Programa que lee un numero complejo y obtiene su forma polar**\n");
  p("\n Introduce la parte real del número complejo: \n");
  s("%i",&p1.a);
  p("\n Introduce la parte imaginaria del número complejo: \n");
  s("%f",&p1.b);
  r=sqrt(p1.a*p1.a+p1.b*p1.b);
  A=p1.b/p1.a;
  A=atan(A);
  p("\n r es %.3f\n",r);
  p("\n A es %.3f\n",A); return 0;}
```

Ejem9

```
#include<stdio.h>
#include<math.h>
#define p printf
#define s scanf

int main(){
 struct{ //local
      int a;
      float b;
 }polar;
 float r, A;
 p("\t **Programa que lee un numero complejo y obtiene su forma polar**\n");
```

```
  p("\n Introduce la parte real del numero complejo: \n");
  s("%i",&polar.a);
  p("\n Introduce la parte imaginaria del numero complejo: \n");
  s("%f",&polar.b);
  r=sqrt(polar.a*polar.a+polar.b*polar.b);
  A=polar.b/polar.a;
  A=atan(A);
  p("\n r es %.3f\n",r);
  p("\n A es %.3f\n",A);
return 0;
}
```

Archivos

Lectura y Escritura de Datos

Los archivos de datos se utilizan cuando el volumen de datos es significativo. En la actualidad, prácticamente todas las aplicaciones requieren almacenar datos en un archivo; por ejemplo, las aplicaciones de los bancos, casas de bolsa, líneas aéreas para la reservación de vuelos y asientos, hospitales, hoteles, escuelas, etc.

El formato de los archivos generalmente es de texto o binario y la forma de acceso a los mismos es secuencial o de acceso directo. En los primeros lenguajes de alto nivel como Pascal existía prácticamente una relación directa entre el formato del archivo y el método de acceso utilizado. Sin embargo, las cosas han cambiado con el tiempo, en el lenguaje C, existen funciones que permiten trabajar con métodos de acceso directo aun cuando el archivo tenga un formato tipo texto.

En los archivos de texto los datos se almacenan en formato texto y ocupan posiciones consecutivas en el dispositivo de almacenamiento secundario. La única forma de acceder a los componentes de un archivo de texto es hacerlo en forma secuencial. Es decir, accediendo al primer componente, luego al segundo, y así sucesivamente hasta llegar al último, y por consiguiente al fin del archivo. Un elemento importante cuando se trabaja con archivos de texto es el área del buffer, que es el lugar donde los datos se almacenan

temporalmente mientras se transfieren de la memoria al dispositivo secundario en que se encuentran o viceversa.

El lenguaje de programación C no impone restricciones ni formatos específicos para almacenar elementos en un archivo. Además, proporciona un conjunto extenso de funciones de biblioteca para el manejo de archivos. ***Es importante señalar que antes de trabajar con un archivo debemos abrirlo y cuando terminamos de trabajar con él debemos cerrarlo por seguridad de la información que se haya almacenado.***

En el lenguaje C un archivo básicamente se abre y cierra de la siguiente forma:

```
/* El conjunto de instrucciones muestra la sintaxis para abrir y cerrar
un archivo en  el lenguaje de programación C. */
. . .
FILE *apuntador_archivo;
apuntador_archivo = fopen (nombre_archivo, "tipo_archivo");
if (apuntador_archivo != NULL)
{
proceso; /* trabajo con el archivo. */
fclose(apuntador_archivo);
}
```

```
else
printf("No se puede abrir el archivo");
. . .
```

La primera instrucción:

```
FILE *apuntador_archivo;
```

Indica que apuntador_archivo es un apuntador al inicio de la estructura FILE, área del buffer que siempre se escribe con mayúsculas.

Un apuntador a un archivo es un hilo común que unifica el sistema de e/s con un buffer donde se transportan los datos. Señala la información que contiene y define ciertas características sobre él, nombre, estado, posición actual del archivo, etc.

Los apuntadores a un archivo se manejan en C como variables de tipo apuntador FILE que define la librería stdio.h

La segunda instrucción:

apuntador_archivo = fopen (nombre_archivo, "tipo-archivo");

Permite abrir un archivo llamado nombre_archivo que puede ser una variable de tipo cadena de caracteres, o bien, una constante sin extensión o con extensión txt para realizar actividades de tipo_archivo.

La función fopen tiene dos argumentos:

- nombre del archivo
- tipo de archivo, que puede ser de lectura, escritura, etc.

En la siguiente tabla se muestran los diferentes tipos de archivos.- modos-

Tipo de archivo –modos-

"r" Se abre un archivo sólo para lectura.

"w" Se abre un archivo sólo para escritura. Si el archivo ya existe, el apuntador se coloca al inicio y sobrescribe, destruyendo al archivo anterior.

"a" Se abre un archivo para agregar nuevos datos al final. Si el archivo no existe, crea uno nuevo.

"r+" Se abre un archivo para realizar modificaciones. Permite leer y escribir. El archivo tiene que existir.

"w+" Se abre un archivo para leer y escribir. Si el archivo existe, el apuntador se coloca al inicio, sobrescribe y destruye el archivo anterior.

"a+" Se abre un archivo para lectura y para incorporar nuevos datos al final. Si el archivo no existe, se crea uno nuevo

La instrucción:

if (apuntador_archivo != NULL)

Permite evaluar el contenido del apuntador. Si éste es igual a NULL, implica que el archivo no se pudo abrir, en cuyo caso es conveniente escribir un mensaje para notificar esta situación. Por otra parte, si el contenido del apuntador es distinto de NULL entonces se comienza a trabajar sobre el archivo.

La instrucción:

fclose (apuntador_archivo);

Se utiliza para cerrar el archivo.

Funciones fscanf y fprintf

Estas funciones se comportan como las funciones de entrada y salida que hemos venido manejando en el curso, scanf y printf respectivamente, solo que operan sobre archivos:

Ejemplos

```
fscanf(fileApuntador,"%d",&M[i][j]); //lectura desde un archivo
fprintf(fileApuntador,"%d\t",M[i][j]) ;//escritura en un archivo
```

Ejem./*Leer desde un archivo la matriz M*/

```
#include<stdio.h>
int main(){
    FILE *fileApuntador;
    int M[3][3],i,j;
    fileApuntador=fopen("m1.txt","r");
// El archivo m1.txt, debe exitir en la misma ruta donde está el programa almacenado y debe contener 9 elementos de tipo entero
    if (fileApuntador!= NULL)
     {
        for (i=0; i<3;i++){
        for (j=0; j<3;j++){
            fscanf(fileApuntador,"%d",&M[i][j]);
            }    }
```

```
        fclose(fileApuntador);
        for(i=0;i<3;i++){
            for(j=0;j<3;j++){
             printf("%d\t",M[i][j]);
         }
            printf("\n");
     }   }
   else
     printf("No se puede abrir el archivo");
return 0;
}
```

Ejem./*Crear un archivo que contenga los valores de M*/

```
#include <stdio.h>

int main(){

    FILE *fA;

    int M[3][3],i,j,a;

    fA=fopen("m2.txt","w");

//El archivo m2.txt,  no debería existir, se generá un archivo Nuevo, en caso que exista se sobreescribe,

//el programa  generá el archivo  en la  ruta donde está el programa

    for(i=0;i<3;i++){

        for(j=0;j<3;j++){

            scanf("%d", &M[i][j]);
```

```
        }    }
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            fprintf(fA,"%d\t",M[i][j]);
        }
     fprintf(fA,"\n");
    }
    fclose(fA);
    return 0;
}
```

Investigar las siguientes funciones:

putchar();

getchar();

gets();

puts();

fgets();

fputs();

Ejem./*Crear un archivo de tipo texto*/

```
#include<stdio.h>
int main(){
    FILE *fA;
    char c;
    int salto=0;
    fA=fopen("Texto.txt","w");
    //después cambie a modo “a”
//El archivo Texto.txt, no debe existir, el programa generá el archivo en la ruta donde está el programa
    while(salto<2){
        c=getchar();
        fprintf(fA,"%c",c);
```

```
            if(c=='\n'){
                salto++;
            }
            else{
                salto=0;
            }
        }
        fclose(fA);
return 0;
}
```